POS API User Manual

Table of Contents

Өөрчлөлт хийсэн түүх

Огноо	Албан тушаалтан	Тайлбар
2015-12-01	B.Nasanjargal, D.Ochirpurev, S.Battulga	PosAPI 2.1 First draft of the user manual.
2015-12-07	B.Nasanjargal, D.Ochirpurev, S.Battulga	PosAPI 2.1 Changes to field descriptions
2015-12-09	B.Nasanjargal, D.Ochirpurev, S.Battulga	About editing receipts

This translation is not official!!!

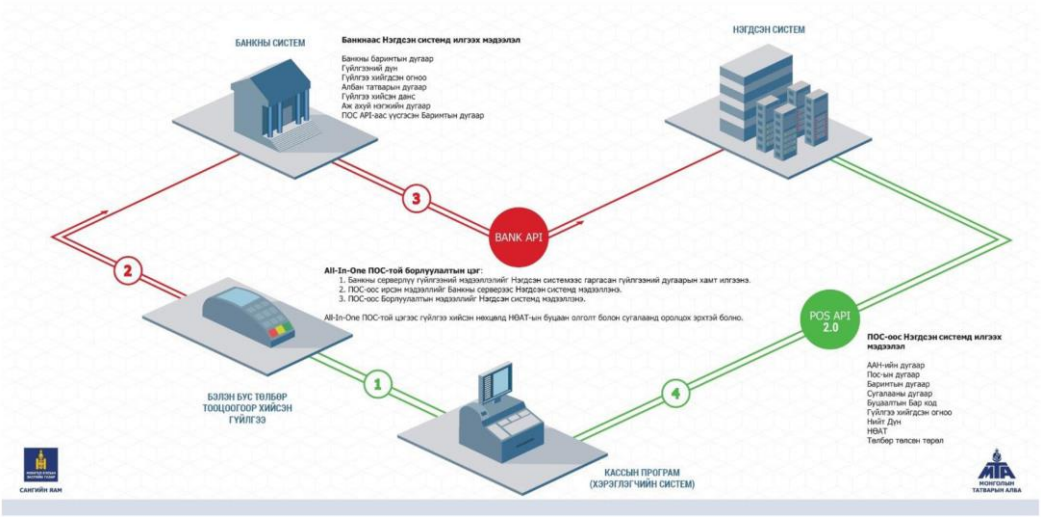
1. Introduction

In regards to the revision of VAT law which is adopted on July 9th of 2015, Pos API 2.1 library has been developed by General Department of Taxation for the purpose of collecting sales receipts of goods and services.

Pos API 2.1 is a software module that is meant to work alongside Cashier Pos system, which collects sales information on goods and services that are being sold to consumers by business entities and individuals.

1.1 Basic principles of Pos API 2.1

How Pos API 2.1 works is shown below:

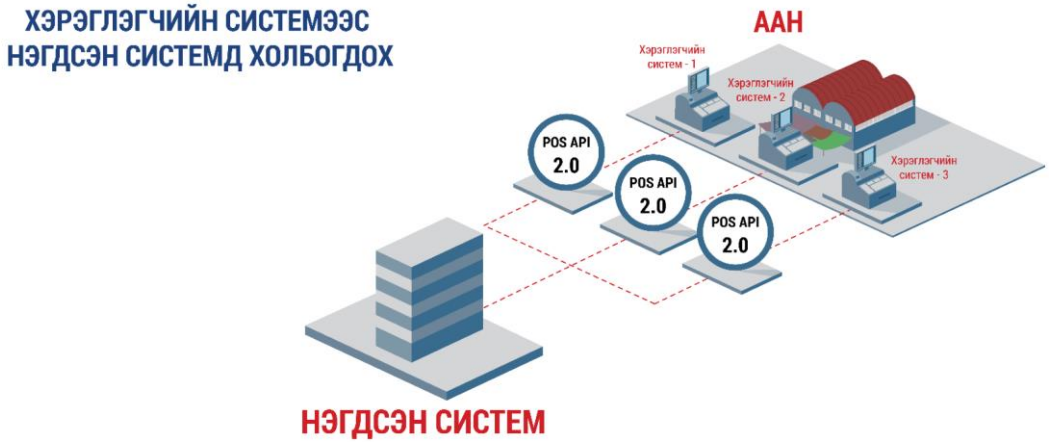


Graph 1: Pos API 2.1

1.2 Scope

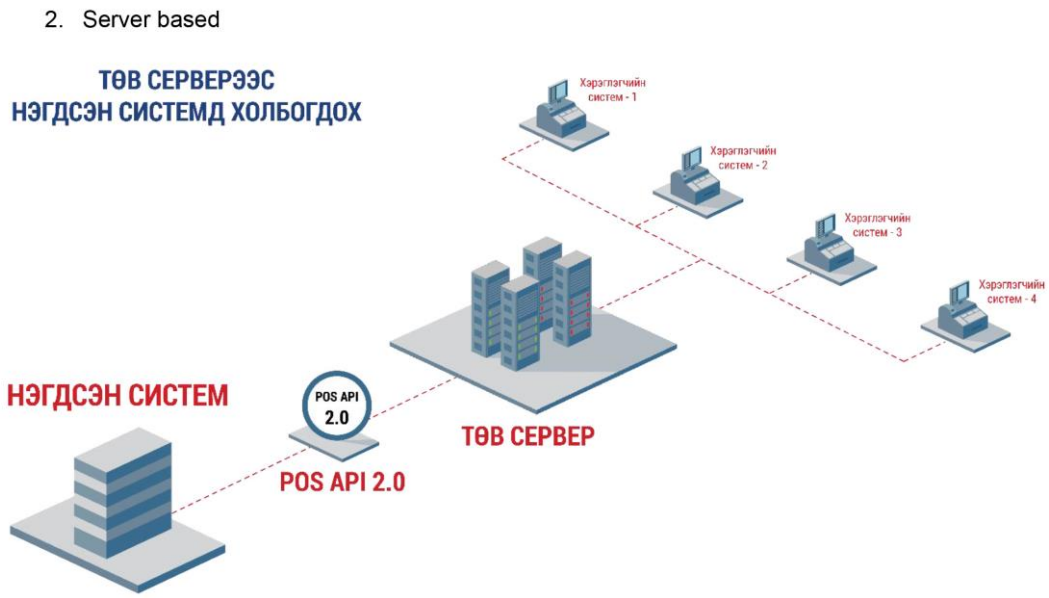
- Value Added Tax Payers
- Capital City Tax Payers

1.3 Client based



Graph 2: Client based operation model

1.4 Server based



Graph 3: Server based operation model

Business entities could use one Pos API 2.1 library to send data to multiple client systems via their central server. Depending on their business characteristics (number of branches, type of services they offer etc.), business entities could choose between the “server side” model (*graph 3*) and the “client side” model (*graph 2*).

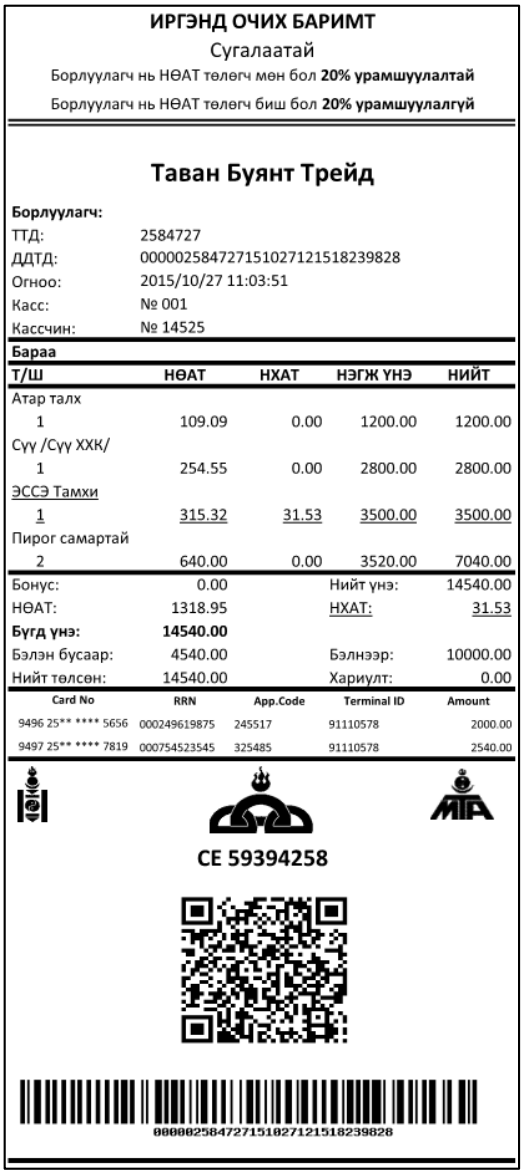
1.5 System Requirements

Minimum system requirements for Pos API 2.1 are:

- Processor 900 MHz or more
- RAM 512 MB or more
- HARD Disk 50 GB of free space
- Graphics DirectX 9 or more

1.6 Electron Receipt

According to the VAT Law, electron receipt will be the foundation block of lottery and promotion allocation as well as the basis of VAT invoice. Therefore, PosAPI will register electron receipts as “B2C” (Business to consumer) and “B2B” (Business to business) and will produce the appropriate data for each one.



ИРГЭНД ОЧИХ БАРИМТ
Сугалаатай
Борлуулагч нь НӨАТ төлөгч мөн бол **20% урамшуулалтай**
Борлуулагч нь НӨАТ төлөгч биш бол **20% урамшуулалгүй**

Таван Буянт Трейд

Борлуулагч:
ТТД: 2584727
ДДТД: 00000258472715102712151823982 8
Огноо: 2015/10/27 11:03:51
Касс: № 001
Кассчин: № 14525

Бараа

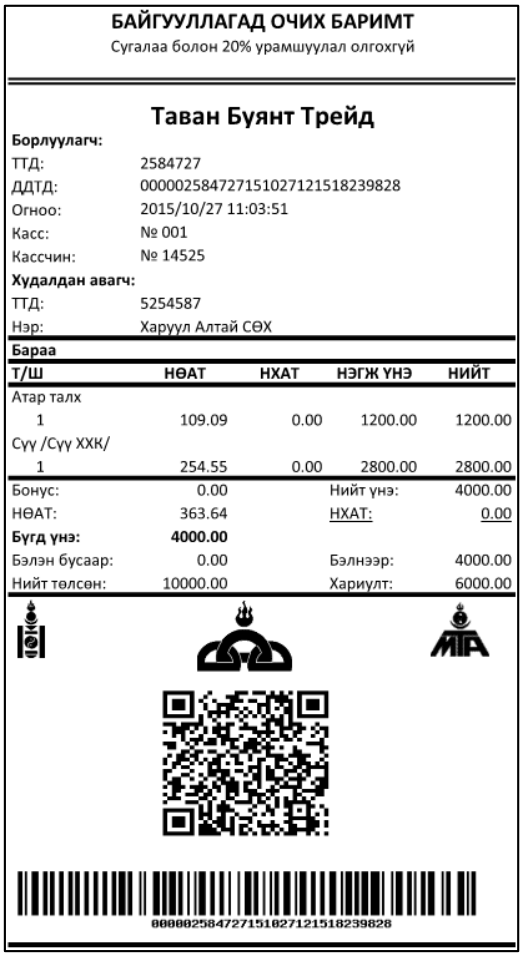
Т/Ш	НӨАТ	НХАТ	НЭГЖ ҮНЭ	НИЙТ
Атар талх				
1	109.09	0.00	1200.00	1200.00
Сүү /Сүү ХХК/				
1	254.55	0.00	2800.00	2800.00
ЭССЭ Тамхи				
1	315.32	31.53	3500.00	3500.00
Пирог самартай				
2	640.00	0.00	3520.00	7040.00

Бонус:	0.00	Нийт үнэ:	14540.00
НӨАТ:	1318.95	НХАТ:	31.53
Бүгд үнэ:	**14540.00**		
Бэлэн бусаар:	4540.00	Бэлнээр:	10000.00
Нийт төлсөн:	14540.00	Хариулт:	0.00

Card No	RRN	App.Code	Terminal ID	Amount
9496 25** **** 5656	000249619875	245517	91110578	2000.00
9497 25** **** 7819	000754523545	325485	91110578	2540.00

CE 59394258

00000258472715102712151823982 8



БАЙГУУЛЛАГАД ОЧИХ БАРИМТ
Сугалаа болон 20% урамшуулал олгохгүй

Таван Буянт Трейд

Борлуулагч:
ТТД: 2584727
ДДТД: 00000258472715102712151823982 8
Огноо: 2015/10/27 11:03:51
Касс: № 001
Кассчин: № 14525

Худалдан авагч:
ТТД: 5254587
Нэр: Харуул Алтай СӨХ

Бараа

Т/Ш	НӨАТ	НХАТ	НЭГЖ ҮНЭ	НИЙТ
Атар талх				
1	109.09	0.00	1200.00	1200.00
Сүү /Сүү ХХК/				
1	254.55	0.00	2800.00	2800.00

Бонус:	0.00	Нийт үнэ:	4000.00
НӨАТ:	363.64	НХАТ:	0.00
Бүгд үнэ:	**4000.00**		
Бэлэн бусаар:	0.00	Бэлнээр:	4000.00
Нийт төлсөн:	10000.00	Хариулт:	6000.00

00000258472715102712151823982 8

Above images of electron receipts are only exemplary and they don't necessarily have to be replicated by other business entities.

1.7 E-receipt requirements

E-receipts produced by business entities should follow the "General requirements of technical devices to be connected for Tax system" /MNS 5005:2015/ standard developed by Agency of Standardization and Metrology (MASM).

Required fields

Locations	Field	For consumer	For business
Top side of the receipt	Merchant tax payer number	Yes	Yes
	Merchant name	Yes	Yes
	Consumer tax payer number	No	Yes
	Consumer name	No	Yes
	Irreplicable payment number	Yes	Yes
	Date	Yes	Yes
Lower side of the receipt	Lottery number	Yes	No
	QR code	Yes	Yes
	Return BarCode	No	No

Even though the return BarCode is not required it would be helpful to have in case of e-receipt cancellation process which would require the Irreplicable number (33 digits long) to be typed by hand.

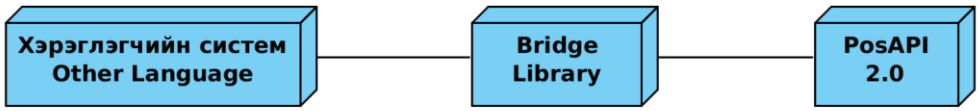
2. PosAPI integration instructions

2.1 Integration methodology

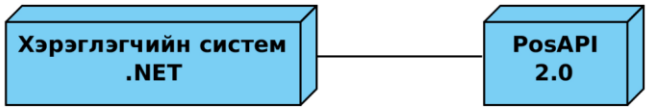
PosAPI2.0 library is written in C++ and to integrate the module with user system, PosAPI.h and ExportLib.h header files should be used. If your Cashier Pos development environment is on .NET framework, you can use the API without the header file.

Even though C++ is a platform independent language, the compilation output requires a bridge library to be used to integrate depending on the platform. Integration examples on how to integrate with Java and C# programming languages are included at the end of this manual.

C++ programs can work with other languages provided there is a bridge library created. Bridge library is unnecessary when it comes to Windows OS environment programming languages like .NET framework or Delphi.



Зураг 6



Зураг 7

2.2 Installation and configuration

Qt5 and OpenSSL 1.0 technologies were used for the development of PosAPI 2.1 and some components of these libraries are necessary for the integration of the module. They can be downloaded from https://ebarimt.mn/posApi website.

2.3 Windows installation

Download link: https://slproweb.com/products/Win32OpenSSL.html

Necessary libraries:

- icudt53.dll
- icuin53.dll
- icuuc53.dll
- Qt5Core.dll
- Qt5Network.dll
- Qt5Script.dll
- Qt5Sql.dll
- libeay32.dll
- ssleay32.dll
- sqldrivers/qsqlite.dll

DLL files above has to be copied to the "Working Directory" of your Cashier Pos system. If your system was written in Java, the files can be copied to "**C:\Windows\Sun\Java\Bin**". The directory ("**C:\Windows\Sun\Java\ Bin\platforms**") can be created manually if it does not exist as shown.

Installing "Visual C++ Redistributable Packager for Visual Studio 2013" is required in order to run Pos API 2.1 module on Windows OS because PosAPI dll and other necessary libraries are written using Visual Studio 2013 C++ language.

If you Cashier Pos software is written in Java language, OpenSSL v1.0.2d Light must be installed for your system to work on Windows environment. If your operating system is Windows XP, it has to be upgraded to Windows XP SP3.

OpenSSL download address: https://slproweb.com/products/Win32OpenSSL.html

Example: DLL files copied to working directory:

Name	Date modified	Type	Size
sqldrivers	2015/11/29 15:02	File folder	
DemoPos.exe	2015/11/29 07:36	Application	25 KB
icudt53.dll	2014/9/3 16:42	Application extens...	21,025 KB
icuin53.dll	2014/9/3 16:42	Application extens...	2,412 KB
icuuc53.dll	2014/9/3 16:42	Application extens...	1,675 KB
libeay32.dll	2015/11/29 21:21	Application extens...	1,611 KB
PosAPI.dll	2015/11/29 14:59	Application extens...	116 KB
Qt5Core.dll	2015/8/21 16:29	Application extens...	4,825 KB
Qt5Network.dll	2015/5/31 02:24	Application extens...	1,017 KB
Qt5Script.dll	2015/5/31 05:39	Application extens...	1,218 KB
Qt5Sql.dll	2015/5/31 02:24	Application extens...	196 KB
ssleay32.dll	2015/11/29 21:21	Application extens...	335 KB

Graph 8

In the above picture, if DemoPos.exe is assumed as the executable of Cashier Pos software and the directory that the file exists in is called the Working Directory. By copying the necessary DLL files in the directory, the executable can find and work with the DLL files directly.

2.4 Linux installation

Necessary libraries:

- libicudata.so.53
- libicuuc.so.53
- libicui18n.so.53
- libQt5Core.so.5.4.1
- libQt5Network.so.5.4.1
- libQt5Script.so.5.4.1
- libQt5Sql.so.5.4.1
- libcrypto.so.1.0.0
- libssl.so.1.0.0

As for Linux OS, above mentioned libraries should be copied to "**/usr/lib**" directory along with the PosAPI (libPosAPI.so). OpenSSL library comes installed already on most Linux distributions so it is not required to install.

3. PosAPI methods

All PosAPI methods use JSON strings to input and output data. Usage of JSON reduces the risk of data type incompatibility errors during transmission between softwares and the programming languages and also advantageous in a sense that JSON is the most widely used format among programming languages.

3.1 UString datatype

Windows and Linux operating systems each have their own ways of working with Unicode string therefore strings are received differently depending on the OS.

"**UString**" type converts into std::wstring on Windows but it converts into std::string on Linux.

3.2 UString PosAPI::checkAPI()

To ensure the stability of the Cashier Pos system, this method performs an operation check on PosAPI. There must be a home directory in the OS that running the Cashier Pos system, otherwise the method will return error.

Input

None

Output

Output format

```
{
   "success":boolean,
   "database":{
      "success":boolean,
      "message":String
   },
   "config":{
      "success":boolean,
      "message":String
   },
   "network":{
      "success":boolean,
      "message":String
   }
}
```

Output field description

Name	Description
success	Indicates if PosAPI library is working without any issues *true – no problems* *false – potential problems*
database	Indicates if there is any issue with database connection.
success	*true – no problems* *false - potential problems*
message	Return an error message if there is an error
config	Determines if configurations are downloaded and configured properly
success	*true - no problems* *false - potential problems*
message	Return an error message if there is an error
network	Checks the network and internet connection by connecting to the Tax system
success	*true - no problems* *false - potential problems*
message	Return an error message if there is an error

3.3 UString PosAPI::getInformation()

There is a need to check the PosAPI information in case there are multiple APIs in use by the Cashier Pos system. This method is used to obtain the PosAPI information.

Input

None

Output

Output format

```
{
    "registerNo":String,
    "branchNo":String,
    "posId":String,
    "dbDirPath":String,
    "extraInfo":{
        "countBill”: String
    }
}
```

Output field description

Name	Description
registerNo	Tax payer number of the PosAPI user. 7 digit number for business entities, 12 digit number for consumers.
branchNo	Tax office branch number of the PosAPI user 3 digit number /000, 001, 142 etc/
posId	PosAPI ID number
dbDirPath	SQLite database directory path
extraInfo	Extra information provided by PosAPI
countBill	Unsent receipt count

3.4 UString PosAPI::callFunction(UString, UString)

There will be additional methods added to PosAPI in the future and it is not necessary to download the PosAPI again each time. Those additional methods can be called using this function.

Whenever a function is added to the library, it will be announced via our website and will be included in future iterations of this manual.

Input

1st input:

Name of the function to be called

Input format

String /Name of the function/

2nd input:

A parameter to be passed

Input format

JSON String /Different depending on the function to be called/

Output

Output format

JSON String / Different depending on the function to be called /

3.5 Static Ustring PosAPI::put(UString param) method

"put" method receives sales information of goods and service from Cashier Pos in JSON format and sends it back with receipt Irreplicable number, lottery number, date of the printout and QRCode.

From these fields, only Irreplicable number and the date of printout are allowed to be saved on the Cashier Pos system. In addition, to register a receipt, the cashier machine has to have a network device (LAN card). If there is no network device or if the network device is in disabled state, it is not possible to print out a receipt.

Can be disconnected from a network.

Input

Input format

```
{
  "amount": String,
  "vat": String,
  "cashAmount": String,
  "nonCashAmount": String,
  "cityTax": String,
  "districtCode": String,
  "posNo": String,
  "customerNo": String,
  "billType":String,
  "billIdSuffix":String,
  "returnBillId":String,
  "districtCode":String,
  "stocks": [
    ...
    {
      "code": String,
      "name": String,
      "measureUnit": String,
      "qty": String,
      "unitPrice": String,
      "totalAmount": String,
      "cityTax": String,
      "vat": String,
      "barCode": String
    }
    ...
  ],
  "bankTransactions": [
    ...
    {
      "rrn": String,
      "bankId": String,
      "terminalId": String,
      "approvalCode": String,
      "amount": String
    }
    ...
  ]
}
```

Input field description

Нэр	Нөхцөл	Тайлбар
amount	2 place decimal number	Transaction sum amount
vat	2 place decimal number	Transaction VAT amount
cashAmount	2 place decimal number	Cash payment amount
nonCashAmount	2 place decimal number	Non-cash payment amount
cityTax	2 place decimal number	City tax sum amount
districtCode	2 digit integer number	Receipt printout location code /Local tax office code/
posNo	4-6 digit integer number	Internal cashier number of the

		organization
customerNo	7 digit integer number or Mongolian citizen registration number	Customer Tax payer number or consumer registration number
billType	1 digit integer number	Receipt type
billIdSuffix	6 digit integer number	Internal number for the purpose of making receipt irreplicable number unique. Can't be replicated within that day.
returnBillId	33 digit whole number	Irreplicable number of the receipt to be edited
stocks		
code	String	Goods and services code (Business entities' own coding)
name	String	Goods and services name / Business entities' own naming/
measureUnit	String	Measuring unit
qty	2 place decimal number	Quantity
unitPrice	2 place decimal number	Unit price
totalAmount	2 place decimal number	Total amount
cityTax	2 place decimal number	City tax sum amount
vat	2 place decimal number	VAT amount
barCode	Whole number	Goods or services barcode or category code
bankTransactions		
rrn	12 digit integer number	Non-cash transaction receipt number
bankId	2 digit integer number	Pos terminal bank ID
terminalId	6 or more character long string (number and letters both)	Pos terminal number
approvalCode	10 character long string (number and letters both	Non-cash transaction authorization code
amount	2 place decimal number	Non-cash transaction amount

Some of the fields from above table are not required to be filled and should be filled only when the system environment or sales type demands it.

Non-compulsory fields

Name	**Description**
billIdSuffix	Has to be filled when there are multiple sales data incoming to PosAPI simultaneously. Because there is a chance that same irreplicable number being generated when there are more than one sales data incoming.
customerNo	Has to be filled when registering a B2B transaction. Not required in case of B2C.
billType	Has to be filled in case of a B2B transaction.

billIdSuffix field contains an irreplicable number within the business entity or within a branch depending on if the PosAPI had a branch number or not.

Receipt type /billType талбар/

Value	Description
1	B2C **sales** receipt of goods and services
2	B2B **purchase** receipt of goods and services
3	B2B **sales** receipt of goods and services

Банкны код /bankId талбар/

Утга	Харгалзах утга
01	Mongolian bank
02	Capital bank
04	Trade and Development bank
05	KHAAN bank
15	Golomt bank
19	Trans bank
21	Arig bank
22	Credit bank
26	Ulaanbaatar City bank
29	National Investment Bank of Mongolia
30	Capitron bank
32	KHAS bank
33	Chinggis khan bank
34	State bank
36	Development bank of Mongolia
38	Bogd bank

Аймаг/Дүүргийн код /districtId талбар/

Value	Description
01	Arkhangai
02	Bayan-Ölgii
03	Bayankhongor
04	Bulgan
05	Govi-Altai
06	Dornogovi
07	Dornod
08	Dundgovi
09	Zavkhan
10	Ovorkhangai
11	Omnogovi
12	Sukhbaatar
13	Selenge
14	Tov
15	Uvs
16	Khovd
17	Khovsgol
18	Khentii
19	Darkhan-Uul
20	Orkhon
32	Govisumber
23	Khan-Uul
24	Bayanzurkh
25	Sukhbaatar
26	Bayangol
27	Baganuur
28	Bagakhangai
29	Nalaikh
34	Songinokhairkhan
35	Chingeltei

Output

Output format

```
{
    "success":boolean,
    "registerNo":String,
    "billId":String,
    "date":String,
    "macAddress":String,
    "internalCode":String,
    "billType":String,
    "qrData":String,
    "lottery":String,
    "lotteryWarningMsg":String,
     ...
}
```

Output field description

Name	Description
success	Indicates if the registration was successfully or not *true – cancellation successful* *false – cancellation unsuccessful*
registerNo	Tax payer number of the PosAPI owner entity or PosAPI эзэмшигч
billId	Receipt irreplicable number 33 digit long /According to VAT Law/
date	Receipt printed date Format: yyyy-MM-dd hh:mm:ss
macAddress	Physical address of the Cashier Pos machine
internalCode	Receipt text code
billType	Receipt type
qrData	Hidden number value in receipt authorization QR code
lottery	Lottery number
lotteryWarningMsg	Warning message shown when lottery ticket cannot be given
errorCode	Error code if there is one
message	Error text

Along with these fields, the fields from earlier transmission will be incoming as well. Only billid and date fields can be saved as they are used for cancelling the receipt.

Receipt editing:

When editing a receipt or executing a partial return, irreplicable number of the receipt to be edited should be entered into returnBillId field. If a receipt is edited, lottery associated with the receipt would be cancelled and there would be no lottery or qrCode printed out for that particular edited receipt.

3.6 UString PosAPI::returnBill(UString)

"ReturnBill" method is used to register sales receipts of goods and services as "cancelled". When returned using this method, lottery and irreplicable number of the receipt will be deemed "cancelled" as well.

Input

Input format

```
{
    "returnBillId":String,
    "date":String
}
```

Input field description

Name	Condition	Description
returnBillId	33 digit number value	Irreplicable number of the receipt to be cancelled
date	Format: yyyy-MM-dd hh:mm:ss	Receipt printout date

Output

Output format

```
{
    "success":boolean,
    "errorCode":Integer,
    "message":String
}
```

Output

Name	Description
success	Indicates if the cancellation was successful or not *true – cancelled* *false – not cancelled*
errorCode	Error code if there is one
message	Error message

3.7 Ustring PosAPI::sendData() method

According to the revised VAT Law, receipt data must be sent within 72 hours after the sale. "sendData" method is used to send the sales receipts of goods and services to VATPS. In order to execute the method successfully, the Cashier Pos system must be connected to internet.

Also if PosAPI is installed for the first time, this method is called to download lottery package and configurations.

If you install PosAPI 2.0 database on a Cashier Pos for the first time, you must execute sendData method without any actual data. By doing so, you register the PosAPI on your machine in VATPS and download required setup /configuration/ information.

Input

None

Output

Output format

```
{
    "success":boolean,
    "errorCode":Integer,
    "message":String
}
```

Output field description

Name	Description
success	Indicates if the data was sent successfully or not. *true - successful* *false - unsuccessful*
errorCode	Error code if there is one
message	Error message

4. Error message description

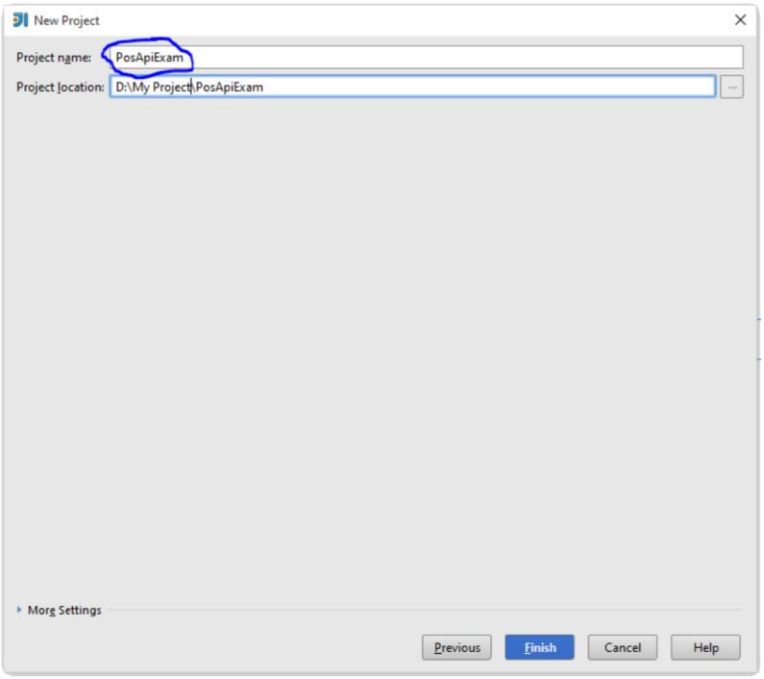
Code	Message	Description
1000	System error	Error related to Operating System environment.
1001	System error	
1002	System error	
1003	System error	
1004	System error	
1005	System error	
0	Key Not Found !!!	Error when configuration data is not downloaded, downloaded incorrectly or the configuration data is wrong.
500	Data transfer too slow. Data transfer upper limit is {0} seconds!!!	Network speed too slow or there is packet loss. Initial system data transfer upper limit is 30 seconds.
501	Data transfer too slow. Data transfer upper limit is {0} seconds!!!	
300	Error while creating or opening a database.\n	Error while trying to create a SQLite database file or trying to connect to a database. Often occurs because of a lack of home directory.
301	Error while creating database. (INFO_DATA)\n	
302	Error occurred while working with database!!!\n	Wrong query executed on database
303	Error occurred while working with database!!!\n	
304	Error occurred while working with database!!!\n	
305	Database connection error!!!	Error occurred while connecting to SQLite database
400	Configuration data corrupted!!! :	Error when configuration data is not downloaded, downloaded incorrectly or the configuration data is wrong.
401	Configuration data corrupted!!! :	
402	Configuration data corrupted!!! :	
403	Configuration data corrupted!!! :	
100	Configuration data not found. Please download the configuration data.	Error occurred after downloading PosAPI for the first time and trying to register a payment receipt or trying to cancel a payment receipt. Please follow the error message instructions.
310	Error occurred while working with database!!!\n	Wrong query executed on database
311	Error occurred while working with database!!!\n	
312	Error occurred while working with database!!!\n	
313	Error occurred while working with database!!!\n	
314	Error occurred while working with database!!!\n	
315	Error occurred while working with database!!!\n	

316	Error occurred while working with database!!!\n	
317	Error occurred while working with database!!!\n	
318	Error occurred while working with database!!!\n	
320	Error occurred while working with database!!!\n	
210	Incorrect data filled into field /{0}/. Validation: {1}	Success conditions not met while trying to register a payment receipt. {0} is JSON field name, {1} RegEX validation.
220	Cash amount and non-cash amount does not equal to total amount.	nonCashAmount + cashAmount == amount condition not met.
600	Access unsuccessful!!! \n	Error shown when the PosAPI is banned from Tax system, or faulty network connection when attempt was made.
601	Error: [{0}]	Error shown when trying to send payment receipt data.
2	Configuration file corrupt.	Configuration data not downloaded or changed manually.
1	Configuration file corrupt.	
200	Error while converting JSON	JSON data structure error
201	Error while converting JSON	

5. Integrating with Java

In order to integrate Java and C++, JNI (Java Native Interface) is used. In below example, Intellij IDEA and Visual Studio 2013 are used.

Step 1: Project creation



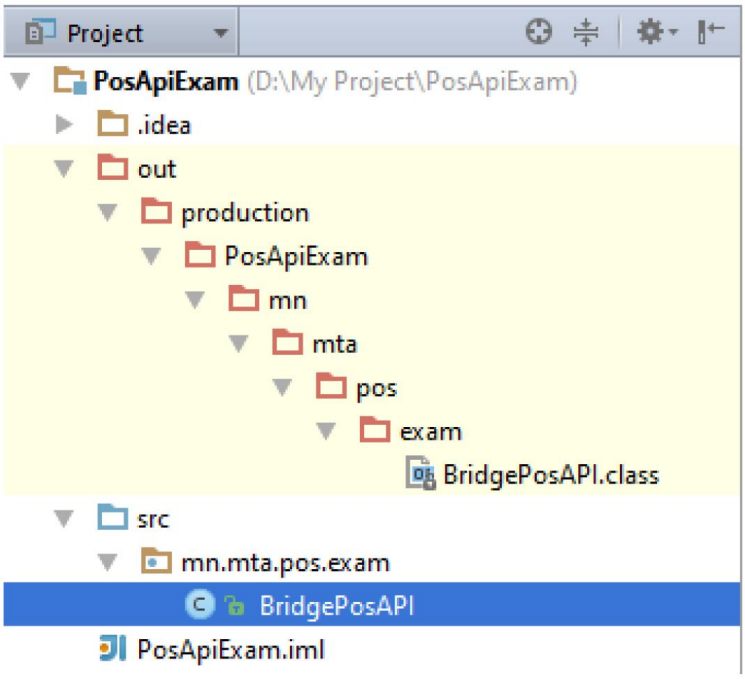
Step 2: Native method creation

```
package mn.mta.pos.exam;

public class BridgePosAPI {
        public static native String put(String data);
        public static native String returnBill(String data);
        public static native String sendData();
        public static native String checkAPI();
        public static native String getInformation();
        public static native String callFunction(String funcName, String data);
}
```

Above methods each call C++ methods: PosAPI::put, PosAPI::returnBill, PosAPI::sendData, PosAPI::checkAPI, PosAPI::getInformation, PosAPI::callFunction respectively.

Creating a C header file from native class

In order to create a C header file, BridgePosAPI class has to be compiled first. To do that you have to press "CTRL+F9" or choose "Build -> Make Project" from the main menu. If the compilation is successful, below screen will be shown:



If the above is shown, open the console (CMD) and execute following commands.

cmd> cd D:\My Project\PosApiExam\out\production\PosApiExam

cmd> javah mn.mta.pos.exam.BridgePosAPI

Purpose of javah command is to create C header file from a java native class.

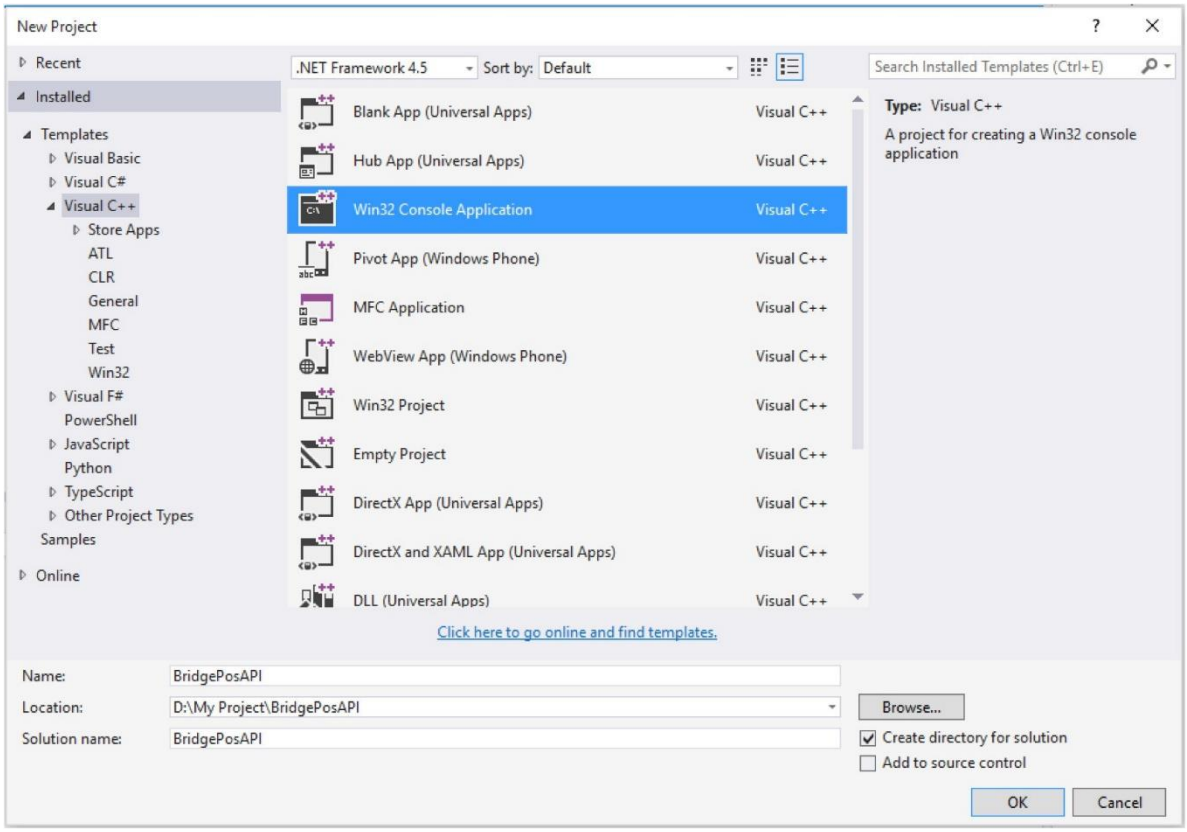
mn_mta_pos_exam_BridgePosAPI.h:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class mn_mta_pos_exam_BridgePosAPI */

#ifndef _Included_mn_mta_pos_exam_BridgePosAPI
#define _Included_mn_mta_pos_exam_BridgePosAPI
#ifdef cplusplus
extern "C" {
#endif
        /*
        *  Class:          mn_mta_pos_exam_BridgePosAPI
        *  Method:         put     * Signature: (Ljava/lang/String;)Ljava/lang/String;
        */
        JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_put
                        (JNIEnv *, jclass, jstring);
        /*
        *  Class:          mn_mta_pos_exam_BridgePosAPI
        *  Method:         returnBill
        * Signature: (Ljava/lang/String;)Ljava/lang/String;
        */
        JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_returnBill
                        (JNIEnv *, jclass, jstring);

        /*
        *  Class:          mn_mta_pos_exam_BridgePosAPI
        *  Method:         sendData     * Signature: ()Ljava/lang/String;
        */
        JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_sendData
                (JNIEnv *, jclass);

        /*
        *  Class:          mn_mta_pos_exam_BridgePosAPI
        *  Method:         checkAPI     * Signature: ()Ljava/lang/String;
        */
        JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_checkAPI
```

```
        (JNIEnv *, jclass);

/*
*  Class:         mn_mta_pos_exam_BridgePosAPI
*  Method:        getInformation          * Signature: ()Ljava/lang/String;
*/
JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_getInformation
        (JNIEnv *, jclass);

/*
*  Class:         mn_mta_pos_exam_BridgePosAPI
*  Method:        callInformation         * Signature: ()Ljava/lang/String;
*/
JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_callInformation
        (JNIEnv *, jclass, jstring, jstring);

#ifdef   cplusplus
}
#endif
#endif
```

Step 3: Creating Visual Studio 2013 C++ project

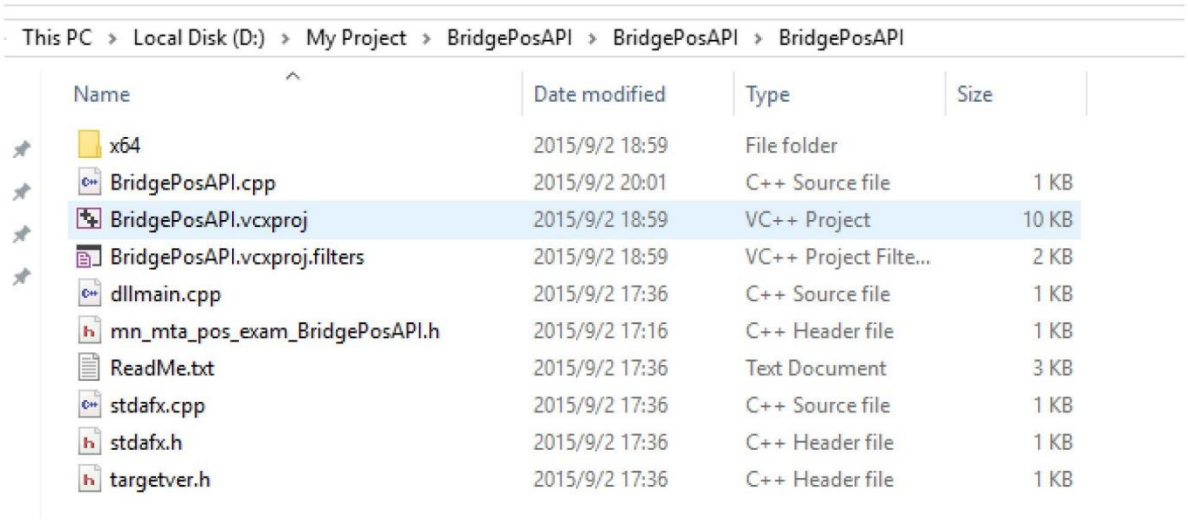
Open Visual Studio 2013 and create the integration library. To create a project, choose "**New Project -> Win32 Console Application**" and press Next.



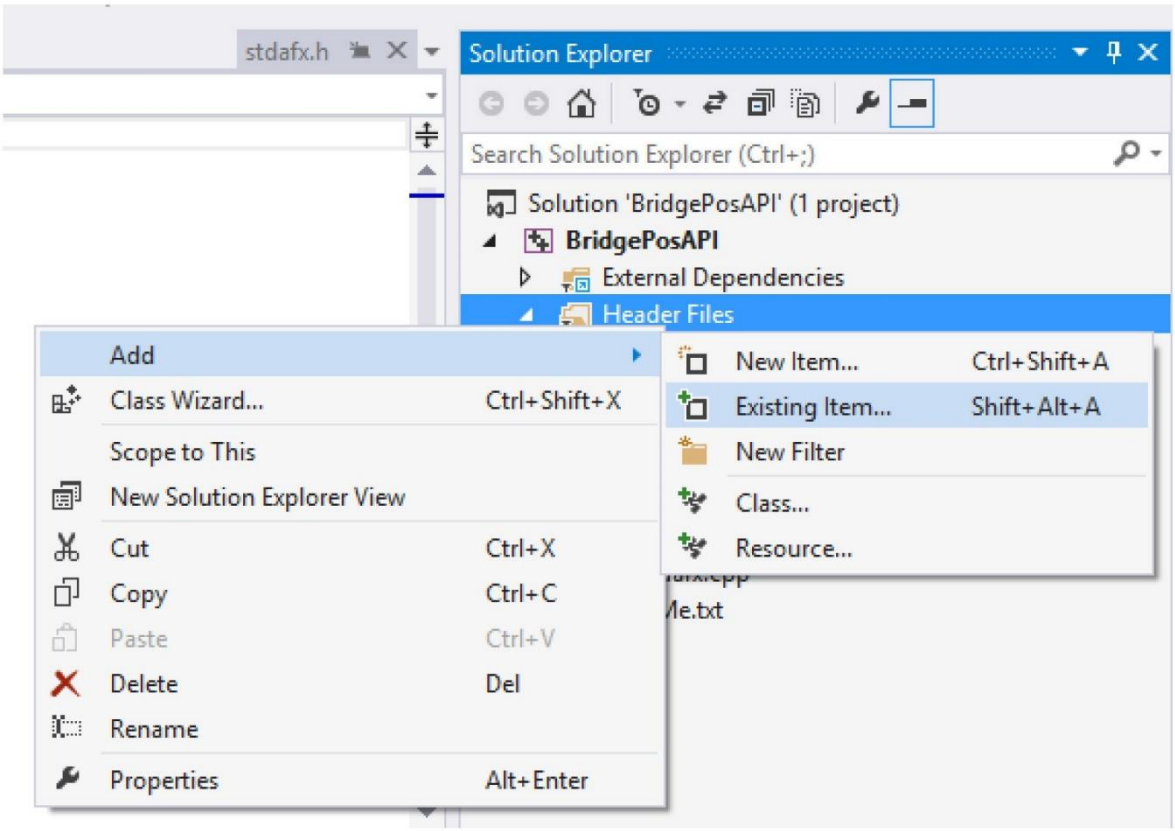
Next, select “Application Type -> DLL” and press Finish.

Step 4: Add mn_mta_pos_exam_BridgePosAPI.h

Copy “mn_mta_pos_exam_BridgePosAPI.h” file that was created earlier to Visual Studio Project directory.



Next, right click on “Solution Explorer -> Header Files” and choose “Add -> Existing Item” to add “mn_mta_pos_exam_BridgePosAPI.h” file.

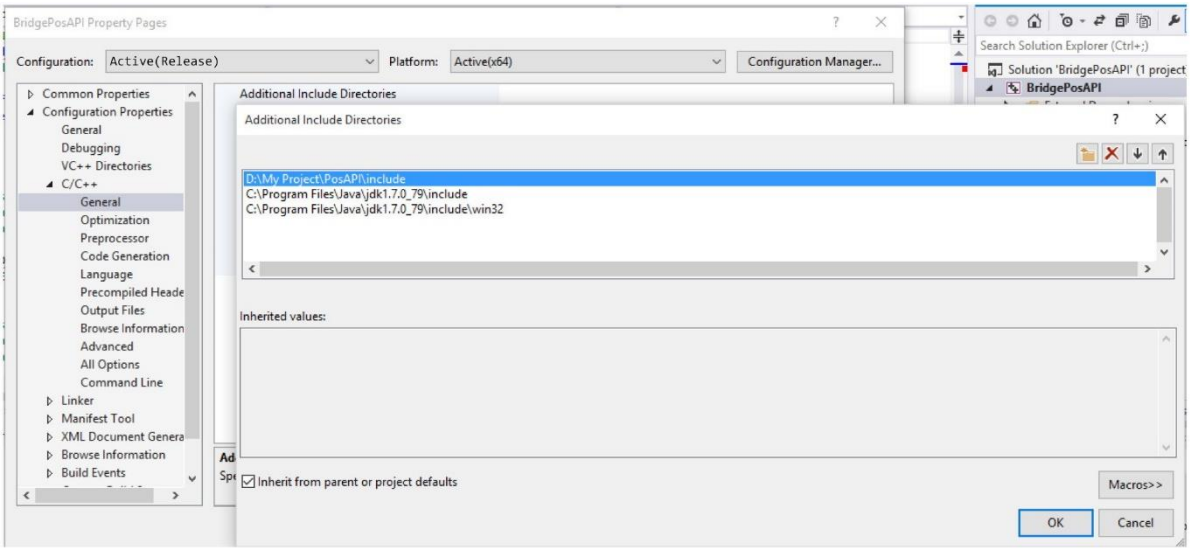


Configuring the integration of JDK header file and PosAPI

Jni.h header file is used when integrating with Java. To add this header file, following steps should be followed.

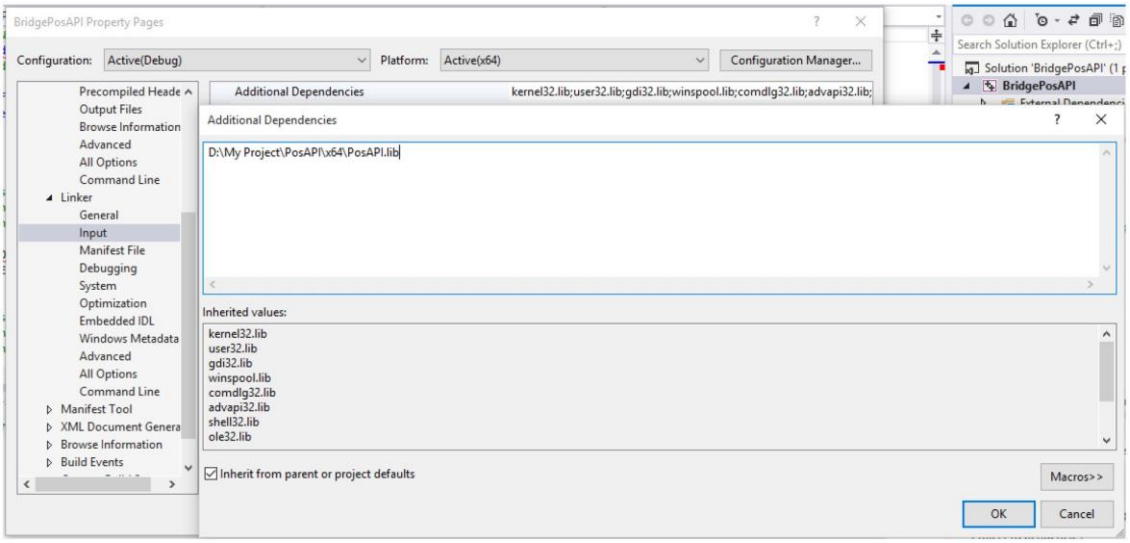
Select “PROJECT -> Property” from the top menu. Next, “Configuration Properties -> C/C++ -> General -> Additional Include Directories” and add following paths.

“[JDK PATH]\include” and “[JDK PATH]\include\win32” paths should be added. Also the download “[PosAPI PATH]\include directory” should be included as well.



After above header files are added, PosAPI.lib file should now be imported.

Select “PROJECT -> Property” from the top menu. Select “Configuration Properties -> Linker -> Input -> Additional Dependencies” and add “[PosAPI PATH]\[Cpu Architecture]\PosAPI.lib” path.



Step 5: Coding the integration in "BridgePosAPI.cpp" file

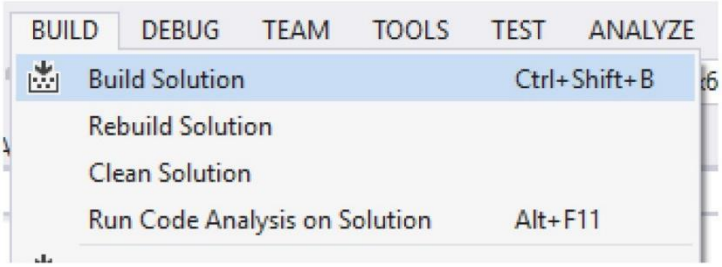
Open BridgePosAPI.cpp file and write the following code.

```
#include "stdafx.h"
#include "mn_mta_pos_exam_BridgePosAPI.h"
#include <PosAPI.h>
#include <codecvt>
#include <locale>
#include <vector>

using namespace vatps;
using namespace std;

/*
* std::string-ийг std::wstring төрөлрүү хөрвүүлэх method
*/
wstring s2ws(const string& str)
{
	typedef codecvt_utf8<wchar_t> convert_typeX;
	wstring_convert<convert_typeX, wchar_t>
	converterX; return converterX.from_bytes(str);
}

/*
* std::wstring-ийг std::string төрөлрүү хөрвүүлэх method
*/
string ws2s(const wstring& wstr)
{
	typedef codecvt_utf8<wchar_t> convert_typeX;
	wstring_convert<convert_typeX, wchar_t>
	converterX; return converterX.to_bytes(wstr);
}

JNIEXPORT jstring JNICALL Java_mn_mta_pos_exam_BridgePosAPI_put (JNIEnv
*env, jclass c, jstring param){
	const char* strParam = env->GetStringUTFChars(param, 0);
	UString data = s2ws(string(strParam));
	UString result = PosAPI::put(data);
	return env->NewStringUTF(ws2s(result).c_str());
}
```

```
JNIEXPORT jstring JNICALL
Java_mn_mta_pos_exam_BridgePosAPI_returnBill
(JNIEnv *env, jclass c, jstring param){
      const char* strParam = env->GetStringUTFChars(param, 0);
      UString data = s2ws(string(strParam));
      UString result =
      PosAPI::returnBill(data);
      return env->NewStringUTF(ws2s(result).c_str());
}

JNIEXPORT jstring JNICALL
Java_mn_mta_pos_exam_BridgePosAPI_sendData
(JNIEnv *env, jclass c){

      UString result = PosAPI::sendData();

      return env->NewStringUTF(ws2s(result).c_str());

}
```

Step 6: Compiling "BridgePosAPI" project and creating dll library

If the code above is successfully written, select "Build -> Build Solution" from the menu and compile the project.



If the compilation is successful, following message will be seen.

1>------ Build started: Project: BridgePosAPI, Configuration: Debug x64 ------

1> BridgePosAPI.cpp

1> Creating library D:\My Project\BridgePosAPI\BridgePosAPI\x64\Debug\BridgePosAPI.lib and object D:\My Project\BridgePosAPI\BridgePosAPI\x64\Debug\BridgePosAPI.exp

1> BridgePosAPI.vcxproj -> **D:\My Project\BridgePosAPI\BridgePosAPI\x64\Debug\BridgePosAPI.dll**

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========

Colored text indicates the location of the newly created dll.

Step 7: Load the library from within Java project

Copy "BridgePosAPI.dll", "PosAPI.dll" libraries along with other secondary libraries to "*C:/Windows/Sun/Java/Bin"* directory. If the directory does not exist, it should be created manually.

After the above steps, libraries should be loaded from within the Java code.

```
package mn.mta.pos.exam;
public class BridgePosAPI {
      static{
            String os = System.getProperty("os.name"); if
               (os.toUpperCase().contains("WINDOWS")) {
                    System.loadLibrary("icudt53");
                    System.loadLibrary("icuuc53");
                    System.loadLibrary("icuin53");
                    System.loadLibrary("Qt5Core");
                     System.loadLibrary("Qt5SQL");
                   System.loadLibrary("Qt5Network");
                   System.loadLibrary("Qt5Gui");
                   System.loadLibrary("Qt5Widgets");
                   System.loadLibrary("PosAPI");
             }
             System.loadLibrary("BridgePosAPI");
      }

      public static native String put(String data);
      public static native String returnBill(String data);
      public static native String sendData();
      public static void main(String[] args) {
             String result = sendData();
             System.out.println("result = " + result);
       }
}
```

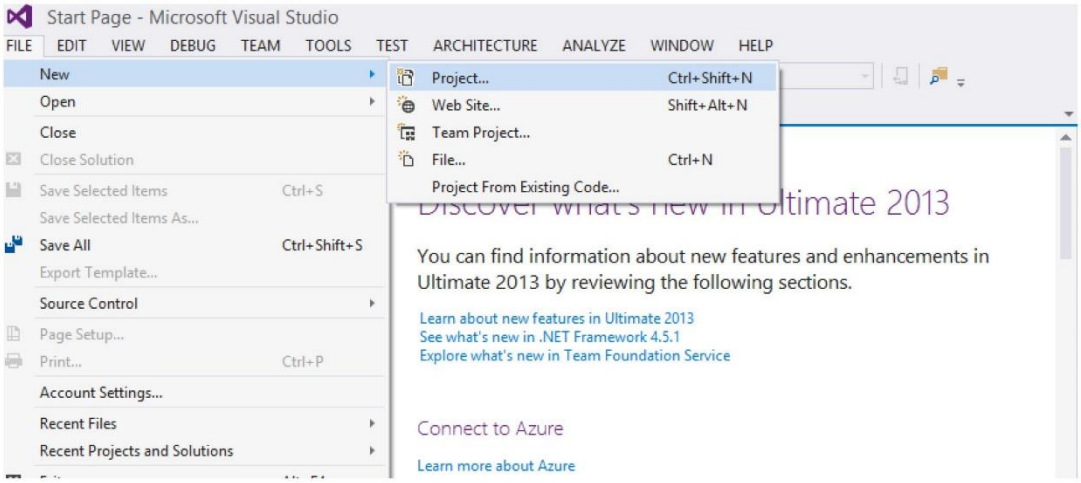
Дээрх кодыг бичсэний дараа main method-ийг дуудахад доорх үр дүн харагдаж байвал таны код амжилттай ажилласан гэж ойлгож болно.
Гаралт:

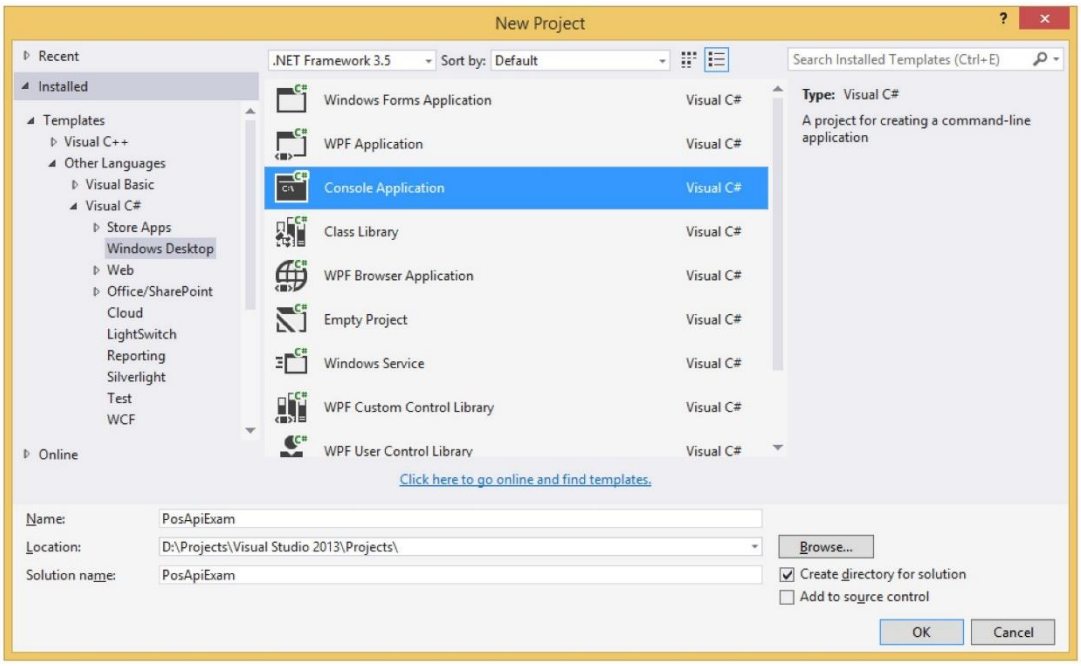
```
result  = {
      "succcess": true
   }
```

6. Integrating with C#

C++ library can be used using C# once Unmanaged C++ library is integrated with C# code. Following example was done in Visual Studio 2013.

Step 1: Project creation





Step 2: Creating Native method

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;

namespace PosApiExam
{
	class Program
	{
		[DllImport(
		“PosAPI.dll”,
		CharSet = CharSet.Unicode, CallingConvention
		= CallingConvention.Cdecl
		)]

		[return: MarshalAs(UnmanagedType.BStr)]
		public static extern string put(String message);

		[DllImport(
		“PosAPI.dll”,
		CharSet = CharSet.Unicode, CallingConvention
		= CallingConvention.Cdecl
		)]

		[return: MarshalAs(UnmanagedType.BStr)]
		public static extern string returnBill(String message);

		[DllImport(“PosAPI.dll”)]
		[return: MarshalAs(UnmanagedType.BStr)]
		public static extern string sendData();

		[STAThread]
		static void Main(string[] args)
		{
			var result = sendData();
			Console.WriteLine(“result = ”+ result);
		}
	}
}
```

Above methods each call C++ mehods: PosAPI::put, PosAPI::returnBill, PosAPI::sendData respectively.

If the following result is returned after writing the above code and invoking the main method, your code has worked successfully.

Output:

```
result = {
        "succcess": true
}
```